LE MATEMATICHE
Vol. LXIII (2008) – Fasc. I, pp. 15–30

SELF-VERIFIED EXTENSION OF AFFINE ARITHMETIC TO ARBITRARY ORDER

GIUSEPPE BILOTTA

Affine Arithmetic (AA) is a self-verifying computational approach that keeps track of first-order correlation between uncertainties in the data and intermediate and final results. In this paper we propose a higher-order extension satisfying the requirements of genericity, arbitrary-order and self-verification, comparing the resulting method with other well-known high-order extensions of AA.

1. Introduction

Self-verifying computing is a way to conduct mathematical operations in such a way that the final result is guaranteed to include the exact (correct) solution of the problem being solved, without any need for a posteriori error analysis. This is attained by modeling data in such a way that it can both represent errors in the initial input data and also keep track of any numerical errors introduced by finite-precision computing and approximate numerical results.

Classical self-verifying methods are based on Interval Arithmetic (IA), initially developed by Moore in 1966 [**?**]. A well-know problem with these methods is the dependency problems, which leads to results wider than necessary when the same variables appear repeatedly in the same expression.

Entrato in redazione 1 ottobre 2007

AMS 2000 Subject Classification: 15A06, 15A15, 15A29, 65G99
Keywords: interval arithmetic, affine arithmetic, dependency problem
Partly sponosered by LIMA Ltd.

For example, a simple subtraction

$$x - x = 0$$

when applied to an interval $x = [a,b]$ with $b \gneq a$ leads to

$$[a,b] - [a,b] = [a,b] + [-b,-a] = [a-b, b-a] \supsetneq 0$$

unless it is known that the two occurences of the interval $[a,b]$ represent the same datum, an information which is often not available in advanced steps of a long computation.

To work around this dependency problem, it is often necessary to rewrite the computations in a way that minimizes repeated occurrences of variables and subexpressions.

Some alternative approaches to self-verifying computing have been devised that reduce the dependency problem, keeping result explosion under control. These methods are often slower step-by-step, and require more memory as they keep track of additional information, but computations often complete in shorter overall time because less steps are needed to obtain tight results. One such method that has been taking traction recently, devised by Stolfi and de Figueiredo in 1997 [**?**] is Affine Arithmetic (AA), an approach that keeps track of first-order dependency in the data and partial results.

AA has been extended to second-order or higher in at least two different ways [**?** **?** **?**]. Each of these extensions has its own benefits, but also some significant shortcomings.

The purpose of this paper is to propose a different way to extend affine arithmetic to arbitrary order by exploiting the benefits of both Modified Affine Arithmetic (MAA) and General Quadratic Forms (GQF) to work around their limitations. Three objectives are met: genericity, arbitrary-order correlation, self-verification.

2. Standard Affine Arithmetic

We will now briefly recall the main operations in standard scalar Affine Arithmetic, as a blueprint for our analysis of its extensions.

2.1. Representation

AA in its standard model represents data as affine expressions

$$x = x_0 + x_1\varepsilon_1 + x_2\varepsilon_2 + \cdots + x_r\varepsilon_r$$

where the x_i are either real numbers (in exact arithmetic) or floating-point numbers (in most numerical applications) and the ε_i are symbolic unknowns whose value ranges independently in $U = [-1,1]$. x_0 is called the *central value*, the x_i for $i > 0$ are called *partial deviations* and the ε_i are called *noise symbols*.

The *radius* or *total deviation* $\mathrm{rad}(x)$ of an affine expression is the radius of the interval of the values spanned as the noise symbols vary in U:

$$\mathrm{rad}(x) = \sum_{i=1}^{r} |x_i|.$$

It is therefore possible to switch from AA to IA by converting an affine expression to interval form (its *range*)

$$x = x_0 + \sum_{i=1}^{r} x_i \varepsilon_i = [x_0 - \mathrm{rad}(x), x_0 + \mathrm{rad}(x)].$$

The purpose of the noise symbols is to keep track of correlation between the data. The ranges of two affine expressions which don't depend on the same noise symbol is independent, and their joint range is a box in the plane. However, when two affine expressions have nonzero coefficients for the same noise symbols, their joint range is constrained to a center-symmetric polytope. For more details on the joint range of affine expressions we refer the reader to [**?**].

It also possible to convert an interval $x = [a,b]$ into an affine expression by considering

$$\mathrm{mid}(x) = \frac{a+b}{2}, \qquad \mathrm{rad}(x) = \frac{b-a}{2},$$
$$x = \mathrm{mid}(x) + \mathrm{rad}(x)\varepsilon_k$$

where $\mathrm{mid}(x)$ and $\mathrm{rad}(x)$ are the center and radius of the interval x. The index k for the noise symbol should be chosen among the free indices, so that ε_k is either a noise symbol for which all other expressions have a zero coefficient, or a new noise symbol altogether.

If a given affine expression is converted to its interval form and then converted back to affine form, the correlation information tracked by noise symbols is lost. Such a conversion should only be used when the range of the datum is needed independently from other results.

2.2. Linear operations

Linear operations act componentwise, so that given two affine expressions

$$x = x_0 + \sum_{i=1}^{r} x_i \varepsilon_i, \qquad y = y_0 + \sum_{i=1}^{r} y_i \varepsilon_i$$

and two sharp scalar values α, β, we have

$$\alpha x + \beta y = \alpha x_0 + \beta y_0 + \sum_{i=1}^{r} (\alpha x_i + \beta y_i)\varepsilon_i.$$

First-order correlation tracked by noise symbols ensures that, in exact arithmetic, AA linear operations automatically obey cancelation laws, a considerable benefit over linear operations in IA. In floating-point arithmetics, roundoff effects may negate complete cancelation, but the results are still much closer to exact zero than in IA.

2.3. Multiplication

As multiplication and inversions are nonlinear functions, the result of their application to affine expressions needs to be approximated by an affine expression: the result will still retain first-order correlation with the input data, but new noise symbols must be introduced to represent higher-order terms.

For the product, consider

$$x = x_0 + \sum_{i=1}^{r} x_i\varepsilon_i, \qquad y = y_0 + \sum_{i=1}^{r} y_i\varepsilon_i;$$

by evaluating their product algebraically we obtain

$$xy = x_0 y_0 + \sum_{i=1}^{r} (x_0 y_i + x_i y_0)\varepsilon_i + R(\varepsilon_1, \ldots, \varepsilon_r)$$

where

$$R(\varepsilon_1, \ldots, \varepsilon_r) = \sum_{i,j=1}^{r} x_i y_j \varepsilon_i \varepsilon_j$$

is of second order in the noise symbols.

If a, b are respectively the minimum and maximum value of R in U^r, the approximation of R with the least error is obviously $\frac{a+b}{2} + \frac{b-a}{2}\varepsilon_k$ (where ε_k is the new noise symbol). However, computation of a and b is expensive, so a quicker approximation using $\operatorname{rad}(x)\operatorname{rad}(y)\varepsilon_k$ is usually preferred.

2.4. On the number of noise symbol

The biggest issue with long standard AA computations with many nonlinear operations is the increasing number of noise symbols: each nonlinear operation needs a new noise symbol, and the same nonlinear subexpression computed at different times will generate a different noise symbol.

Moreover, a new noise symbol needs to be introduced also for each linear operation which cannot be computed exactly in floating-point, to keep track of the roundoff errors.

A strategy to reduce the number of noise symbols is to compact some of the noise terms and replace them with a single noise term whose partial deviation is the sum of the reduced partial deviations. This simplifies the expressions and reduces computational time and memory requirements, at the expense of some loss of correlation.

Commonly, this is done by introducing three noise symbols $\varepsilon_+, \varepsilon_-, \varepsilon_e$ to collect the positive, negative and sign-undefined error terms introduced by roundoff and nonlinear operations ($\varepsilon_+ \in [0,1], \varepsilon_- \in [-1,0], \varepsilon_e \in U$) ([**?** **?**]).

Since the dependency loss of this implementation can lead to significant overestimation of the results, two ways to preserve high-order correlation in nonlinear operation have been devised.

3. General Quatratic Form

General Quadratic Forms (GQF) are an extension of AA that keeps track of second-order noise symbols dependency. If $\varepsilon = (\varepsilon_1, \ldots, \varepsilon_n)$ is the vector of the noise symbols of the original data, then a GQF has the form

$$x = \varepsilon^T A_x \varepsilon + b_x \varepsilon + c_x + e_x^- \varepsilon_- + e_x^+ \varepsilon_+ + e_x \varepsilon_e$$

where A_x is an $n \times n$ matrix, b_x an n-dimensional vector, c_x is a real number and e_x^-, e_x^+, e_x are nonnegative real numbers. Detailed expressions for affine operations and multiplication between GQF are presented in [**?**].

The most significant advantage of GQF over standard AA is that second-order dependecy on the noise symbols is preserved across multiplications, although higher-order terms are still collected in the error terms e_x^- (for the negative terms), e_x^+ (for the positive terms), and e_x (for the sign-undefined terms). In floating-point (non-exact) arithmetic, self-verification of the results can be guaranteed by allowing A_x, b_x, c_x to be interval-valued, thereby accounting for roundoff errors.

Although this extension to standard AA is quite generic and can be satisfactorily used in many cases, it has some significant drawbacks when long chains of multiplications, or high powers of the variables, are involved, since much of the correlation would then be absorbed by the error terms.

Therefore, while it reaches the genericity and self-verification objectives, it doesn't fullly meet the arbitrary-order correlation requirement.

4. Modified AA in tensor form

A radically different extension of affine arithmetic specifically tuned for arbitrary order polynomials is Modified AA in tensor form [**? ?**].

The fundamental idea of MAA in tensor form is that polynomials can be written as a tensor product. Given for example a polynomial in three variables

$$f(x,y,z) = \sum_{i=0}^{n}\sum_{j=0}^{m}\sum_{k=0}^{l} A_{ijk}x^i y^j z^k$$

we can write it as $X \otimes_x (Z \otimes_z A) \otimes_y Y$ where $X = (1,x,\dots,x^n), Y = (1,y,\dots y^m)^T$ and $Z = (1,z,\dots,z^l)$.

Let us assume that the variables are independent and that their affine form is

$$x = x_0 + x_1\varepsilon_x, y = y_0 + y_1\varepsilon_y, z = z_0 + z_1\varepsilon_z.$$

We can then define the power vectors

$$\begin{aligned}\hat{X} &= (1,\varepsilon_x,\dots,\varepsilon_x^n),\\ \hat{Y} &= (1,\varepsilon_y,\dots,\varepsilon_y^n),\\ \hat{Z} &= (1,\varepsilon_z,\dots,\varepsilon_z^n)\end{aligned}$$

and the tensors $C^{(x)}, C^{(y)}, C^{(z)}$ with components

$$\begin{aligned}C^{(x)}_{ij} &= \begin{cases}\binom{j}{i} x_0^{j-i}x_1^i & i \le j,\\ 0 & i > j\end{cases} && i = 0,\dots,n; j = 0,\dots n,\\ C^{(y)}_{ij} &= \begin{cases}0 & i < j,\\ \binom{j}{i} y_0^{j-i}y_1^i & i \ge j,\end{cases} && i = 0,\dots,m; j = 0,\dots m,\\ C^{(z)}_{ij} &= \begin{cases}\binom{j}{i} z_0^{j-i}z_1^i & i \le j,\\ 0 & i > j,\end{cases} && i = 0,\dots,l; j = 0,\dots l,\end{aligned}$$

and use these to write our polynomial in *centered form*

$$f(\varepsilon_z,\varepsilon_y,\varepsilon_z) = \hat{X} \otimes_x (\hat{Z} \otimes_x G) \otimes_y \hat{Y} = \sum_{i=0}^{n}\sum_{j=0}^{m}\sum_{k=0}^{l} G_{ijk}\varepsilon_x^i\varepsilon_y^j\varepsilon_z^k$$

where $G = C^{(x)} \otimes_x (C^{(z)} \otimes_z A) \otimes_y C^{(y)}$.

The total range $[\underline{F}, \overline{F}]$ of the polynomial is then calculated as

$$\underline{F} = G_{000} + \sum_{i,j,k} \begin{cases}\min(0, G_{ijk}) & \text{if } i,j,k \text{ are all even},\\ -|G_{ijk}| & \text{otherwise}\end{cases}$$

and similarly

$$\overline{F} = G_{000} + \sum_{i,j,k} \begin{cases} \max(0, G_{ijk}) & \text{if } i,j,k \text{ are all even,} \\ |G_{ijk}| & \text{otherwise} \end{cases}.$$

The resulting bounds are much tighter than those obtain by standard AA, and slightly better than those obtained by using interval center form computations.

This benefit however comes at the cost of a high specialization: the algorithm cannot indeed be applied in more general situation, when complete *a priori* knowledge of the polynomial expressions involved cannot be obtained. Moreover, the tensor decomposition of a polynomial is only practical when a small number of variables are involved. Finally, possible roundoff errors in the computation of the tensor G are not taken into account.

This extension meets therefore the arbitrary-order correlation requirements, but fails the genericity and self-verification objectives.

5. Extended Polynomial Arithmetic

We will now introduce our arbitrary-order correlation-preserving, generic, self-verified extension to AA, starting from its fundamental building blocks.

5.1. Polynomial terms

For any given symbol x we use X_l^r as a shorthand for

$$\sum x_{i_1 \ldots i_l} \varepsilon_{i_1} \ldots \varepsilon_{i_l}$$

where the summation is for $i_1 = 1 \ldots r$ and $i_j = 1 \ldots i_{j-1}$ for $j > 1$. The superscript r will be omitted where possible. We have thus for example

$$X_0 = x_0, \qquad Y_1 = Y_1^r = \sum_{i=1}^{r} y_i \varepsilon_i,$$

$$Z_2 = Z_2^r = \sum_{i=1}^{r} \sum_{j=1}^{i} z_{ij} \varepsilon_i \varepsilon_j, \qquad W_3 = W_3^r = \sum_{i=1}^{r} \sum_{j=1}^{i} \sum_{k=1}^{j} w_{ijk} \varepsilon_i \varepsilon_j \varepsilon_k.$$

When the coefficients $x_{i_1 \ldots i_l}$ are real numbers and the symbols $\varepsilon_{i_1}, \ldots, \varepsilon_{i_l}$ are unknowns, the expressions represented in this notation will be called *polynomial terms* of order l.

In what follows we will always assume that ε_i are *noise symbols*, i.e. indepent unknowns with range U.

Affine operations on polynomial terms of the same order return a polynomial term of the same order: given two polynomial terms of order l X_l, Y_l and two real numbers α, β we have

$$\alpha X_l + \beta Y_l = \sum (\alpha x_{i_1 \dots i_l} + \beta y_{i_1 \dots i_l}) \varepsilon_{i_1} \dots \varepsilon_{i_l}.$$

The products of two polynomial terms X_l, Y_m of order l, m is a polynomial term Z_{l+m} of order $l+m$. The expression of $z_{i_1, \dots i_{l+m}}$ is given by

$$z_{i_1, \dots i_{l+m}} = \sum x_{j_1, \dots j_l} y_{k_1, \dots k_m}$$

where the summation is extended to all indices $(j_1, \dots, j_l), (k_1, \dots, k_m)$ such that $(j_1, \dots, j_l, k_1, \dots, k_m)$ is a permutation of $(i_1, \dots, i_{l+m})$.

It should be noted that the strict ordering condition on index values limits the number of such combinations: for example, for $l = 2, m = 3$ we have

$$\begin{aligned}
z_{11111} &= x_{11} y_{111}, \\
z_{21111} &= x_{21} y_{111} + x_{11} y_{211}, \\
z_{31111} &= x_{31} y_{111} + x_{11} y_{311}, \\
z_{32111} &= x_{32} y_{111} + x_{31} y_{211} + x_{21} y_{311} + x_{11} y_{321}
\end{aligned}$$

and so on. However, the number of coefficients in a polynomial term X_l^r is given by the number of unordered l-tuples of r elements, i.e. $\binom{r+l-1}{l}$, and products of high-order polynomial terms can become rather expensive.

The *parity* of a coefficient $x_{i_1 \dots i_l}$ of a polynomial term X_l is the parity of the occurences of each unique index value in $(i_1, \dots, i_l)$: if all unique index values occur an even number of times, then the coefficient has even parity, otherwise it has odd parity.

For example x_{332111} has odd parity, because the unique index values 2 and 1 occur an odd number of times. The coefficient x_{332211} has even parity since all of its unique index values $3, 2, 1$ occur an even number of times. All the coefficients of odd-order terms will have odd parity.

The *range* of a polynomial term X_l is the (real-valued) interval $[\underline{X}_l, \overline{X}_l]$ it spans as its noise symbols vary in U. As the order l and the number r of noise symbols grow, however, it becomes computationally very expensive to calculate the actual range, so a faster overestimation is often preferred.

We define the *quick range* $\mathrm{qr}(X_l)$ of a polynomial term X_l in the following way:

- for $l = 0$ we set $\overline{\mathrm{qr}}(X_l) = \underline{\mathrm{qr}}(X_l) = x_0$;
- for l odd we set $\overline{\mathrm{qr}}(X_l) = \sum |x_{i_1 \dots i_l}| = -\underline{\mathrm{qr}}(X_l)$;

- for $l > 0$ even, we set

$$\underline{\mathrm{qr}}(X_l) = \sum_{\text{even}} \min(0, x_{i_1 \ldots i_l}) - \sum_{\text{odd}} |x_{i_1 \ldots i_l}|,$$
$$\overline{\mathrm{qr}}(X_l) = \sum_{\text{even}} \max(0, x_{i_1 \ldots i_l}) + \sum_{\text{odd}} |x_{i_1 \ldots i_l}|$$

  where the summations run separately over even- and odd-parity indices, as indicated. We remark that the overestimation of the quick range over the actual range grows as the order of the polynomial term increases.

5.2. Purely polynomial forms

A *(purely) polynomial form* of degree h with *noise symbols* $\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_r$ is an expression in the form

$$x = \sum_{m=0}^{h} X_m = x_0 + \sum_{i=1}^{r} x_i \varepsilon_i + \sum_{i=1}^{r} \sum_{j=1}^{i} x_{ij} \varepsilon_i \varepsilon_j + \sum_{i=1}^{r} \sum_{j=1}^{i} \sum_{k=1}^{j} x_{ijk} \varepsilon_i \varepsilon_j \varepsilon_k + \ldots$$

To simplify notation in the following formulas, we will introduce the following convention: given a polynomial form of order h $x = \sum_{i=0}^{h} X_i$, we will define X_i for $i > h$ as being identically 0.

Affine operations on polynomial forms follow the usual algebraic rules. Given two polynomial forms $x = \sum_{i=0}^{h} X_i, y = \sum_{j=0}^{k} Y_j$ and two real numbers α, β we have

$$\alpha x + \beta y = \sum_{i=0}^{\max(h,k)} (\alpha X_i + \beta Y_i)$$

The product of two polynomial forms $x = \sum_{i=0}^{h} X_i, y = \sum_{j=0}^{k} Y_j$ can be obtained by formally multiplying their expression, obtaining a new polynomial form $z = \sum_{i=0}^{hk} Z_i$ with $Z_i = \sum_{j=0}^{i} X_j Y_{i-j}$

The *range* of a polynomial form x is the (real-valued) interval $[\underline{x}, \overline{x}]$ it spans as the noise symbols it depends on vary in U. This is however extremely complex to calculate as the order of the polynomial form grows, so it's again necessary to be able to calculate a quicker overstimation. The *quick range* of the polynomial form $x = \sum_{i=0}^{h} X_i$ will be

$$\mathrm{qr}(x) = \sum_{i=0}^{h} \mathrm{qr}(X_i).$$

5.2.1. Order reduction

The computational compexity of multiplying two polynomial forms of degree h,k respectively can be estimated with the following considerations.

The number of products of polynomial terms in such an operation is $l+1$ for the l-th addendum (see above), and thus

$$t = \sum_{l=0}^{hk} (l+1) = 1+2+\cdots+(hk+1) = \frac{(hk+1)(hk+2)}{2}$$

in total; for each of this products we need to calculate $\binom{r+l-1}{l}$ coefficients, totalling to

$$\binom{r+t}{t} = \frac{(r+t)!}{r!t!}.$$

Since t depends quadratically on h,k, the number of operations involved grows much faster with the order of the polynomial terms than with the number r of noise symbols. By contrast, the benefits provided by the higher correlation decrease as the quick range overestimation worsens.

These considerations indicate that usually the order of the expressions involved in a long computation should not be made to grow past a given maximum; this maximum depends on factors such as the width of the uncertainties, the number of uncertainties which are present, and the specific problem to which the method is applied.

When expressions grow past the prescribed maximum order it is therefore necessary to have a method to reduce the order of a given polynomial form; the cost of this reduction is the introduction of a new noise symbol.

Reduction can be achieved in at least two different ways; the method to choose depends chiefly on the absolute value of the coefficients.

Truncation: we say that a polynomial form of order h gets truncated to order $k < h$ if all the polynomial terms of order $j > k$ are replaced by an additional order 1 coefficient. Let $x = \sum_{i=0}^{h} X_i$ be the original polynomial form, and let

$$\chi = \sum_{j=k+1}^{h} \mathrm{qr}(X_i), \qquad c_\chi = \mathrm{mid}(\chi), \qquad r_\chi = \mathrm{rad}(\chi);$$

construct a new polynomial form $y = \sum_{i=0}^{h} Y_i$ by putting

$$Y_0 = X_0 + c_\chi, \qquad Y_1 = X_1 + r_\chi \varepsilon_{r+1}, \qquad Y_i = X_i \; \forall \; i > 1, i \leq k.$$

It is obvious to note that $\mathrm{qr}(y) = \mathrm{qr}(x)$, and that the dependency from $\varepsilon_s, s = 1,\ldots,r$ remains unaltered up to order k. Dependency for higher

orders gets truncated by replacing it with the *truncation interval* χ (in affine form).

This method is to be used in preference when the truncated terms are small in radius, so as to minimize dependency loss.

Power deflation: a more sophisticated strategy consists in the substitution of a product of noise symbols with a first-degree affine expression spanning the same range. Common substitutions are:

$$\begin{aligned}
\varepsilon_s^{2i} &\leftarrow \frac{1}{2} + \frac{1}{2}\varepsilon_{r+1}, \\
\varepsilon_s^{2i-1} &\leftarrow \varepsilon_{r+1}, \\
\varepsilon_s^{2i}\varepsilon_u^{2j} &\leftarrow \frac{1}{2} + \frac{1}{2}\varepsilon_{r+1}, \\
\varepsilon_s^{2i-1}\varepsilon_u^{2j} &\leftarrow \varepsilon_{r+1};
\end{aligned}$$

in general, when all the powers of the noise symbols are even, the substitution uses $\frac{1}{2} + \frac{1}{2}\varepsilon_{r+1}$, otherwise, the new noise symbol is used directly.

It should be noted that a single power deflation does not, in general, reduce the order of the original polynomial form, since it acts on single coefficients, which are shifted from higher order to lower order polynomial terms.

Power deflation is thus preferrably used to shift large-radius high-order coefficients to lower order terms, so that a subsequent truncation step can be used without much loss in dependency.

5.3. Extended polynomial forms

The algebra of purely polynomial forms satisfies our criteria for genericity and arbitrary-order correlation, but does not guarantee self-verification of the results in presence of roundoff errors.

An *extended polynomial form* is an expression

$$x = x_P + x_I$$

where x_P is a purely polynomial form (the *pure* part of the expression) and x_I is the *interval part*, an interval $[\underline{x_I}, \overline{x_I}]$ used to handle the roundoff errors.

Let $z_P = x_P \diamond y_P$ be the result of an operation between the pure parts of two extended polynomial forms, and assume that a coefficient $z_{i_1 \ldots i_l}$ of the result cannot be represented exactly numerically. Assume that $z_{i_1 \ldots i_l} > 0$ and let γ be the largest representable value smaller than $z_{i_1 \ldots i_l}$; also let r be the smallest

representable value such that $\gamma + r \geq z_{i_1 \ldots i_l}$ (the relations are obviously reversed if $z_{i_1 \ldots i_l} < 0$). γ is thefore the chopped (rounded towards zero) value of $z_{i_1 \ldots i_l}$, while r is the representable value of the roundoff.

If the coefficient has odd parity, the absolute value of its roundoff is added to the sign-indefinite roundoff $r_\pm$; if the coefficient has even parity its roundoff is added to the positive r_+ or negative r_- roundoff, depending on whether the coefficient itself is positive or negative. The roundoff interval r_I for the given operation can then be computed as

$$r_I = [-r_\pm + r_-, r_\pm + r_+]$$

and is added to the interval part of z as we will show briefly. Remark that this choices ensures that $0 \in r_I$, guaranteeing that the roundoff correction is always outward.

Let x, y be two extended polynomial forms and let α, β be real numbers; $z = \alpha x + \beta y$ can be computed as follows. For the pure part we have

$$z_P = \alpha x_P + \beta y_P$$

with roundoff interval r_I calculated as described above.

For the interval part we have

$$z_I = \alpha x_I + \beta x_I + r_I$$

where the additions are performed with outer rounding.

The product $z = xy$ is computed as follows. For the pure part we have

$$z_P = x_P y_P$$

with roundoff interval r_I calculated as described above.

For the interval part, let $q_x = \mathrm{qr}(x_P), q_y = \mathrm{qr}(y_P)$ and calculate

$$z_I = q_x y_P + q_y x_P + x_P y_P + r_I$$

where the operations are performed with outer rounding.

The *range* of an extended polynomial form x is the sum of the range of its pure part x_P and the interval part x_I. Once again, it's most common to use the *quick range* which is defined simply as $\mathrm{qr}(x) = \mathrm{qr}(x_P) + x_I$.

We remark here that the choice to round non-representable coefficients inwards (towards zero) serves the purpose of minimizing the range (and quick range) overestimation; indeed, since the range correction introduced by the interval part is always outwards, outer rounding of the coefficients would lead to an unnecessary, although slight, increase in overestimation.

6. Applications

The extended affine arithmetic presented here has been successfully used to improve the outer estimation when evaluating stress in structural analysis of FEM problems with uncertain mechanical parameters.

In mathematical terms, the first step in such a structural analysis is the resolution of a linear system

$$AX = B$$

where

- A is the *stiffness matrix* of the problem, and is built from the mechanical parameters of the elements in which the structure is decomposed; therefore, it can contain uncertainties coming from the uncertainties of the mechanical parameters;
- B is a vector which represents the load which the structure undergoes;
- X is a vector of unknowns representing the displacements of the nodes (vertices of the elements).

In most physical problems, structural analysis leads to the creation of very large linear system: real-world problems can easily have over hundreds of thousands of variables.

It should be noted that in general the number of uncertainties is considerably smaller than the number of variables: in worst-case scenarios where each element is supposed to have mechanical parameters independent from those of the neighbouring elements, for example, one would have an order of magnitude less uncertainties than variables (for example, in a typicial three-dimensional problem with N tethraedral elements one would have $12N$ variables but at most $2N$ uncertainties). In more realistic case, one would consider neighbouring elements to have correlated mechanical parameters and would only consider a handful of noise symbols. [**?**]

One of the most important results in structural analysis is the computed stress which a particular load exherts on the elements of the structure. Stress computation for the elements is obtained from the nodes displacement using both linear and nonlinear operations.

It should be noted that the iterative nature of the solvers used for large sparse linear systems such as those resulting from FEM structural analysis prevent usage of MAA. Therefore, the only meaningful results can be obtained by comparing first (standard AA), second (GQF) and higher order extended affine arithmetic.

The multidimensional nature of the problemi led us to usage of extended affine arithmetic in conjunction with a multidimensional extension to affine arithmetic developed by the author [**?**].

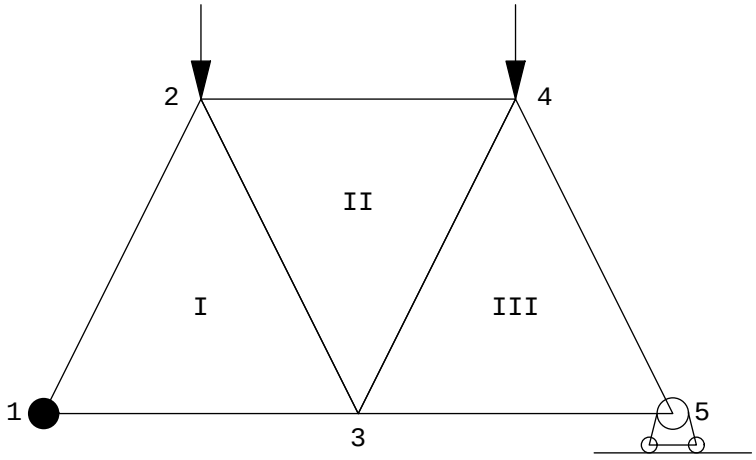

Figure 1: Structural analysis example: initial position

We show the benefits of higher-order affine arithmetic by showing the results for a simple 2D example, depicted in Figure 1. A steel beam (Young's modulus $200N/m^2$, Poisson ratio 0.29) is bound on the leftmost side and can move freely on the horizontal axis. With the prescribed load of $100N$ on the two indicated vertex, the displacement would be as indicated in Figure 2

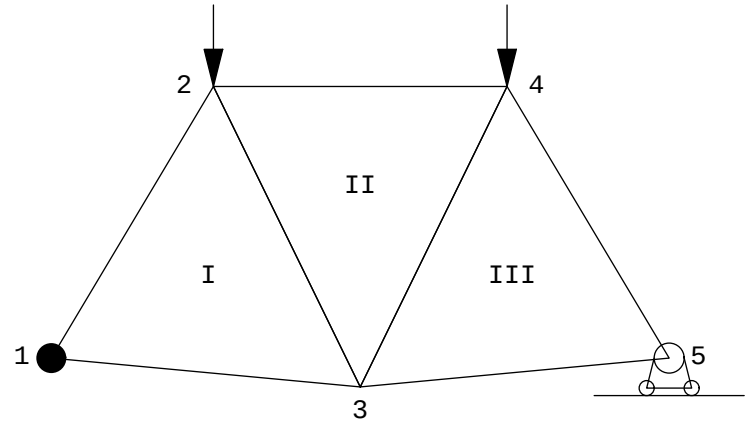

Figure 2: Structural analysis example: result

However, the Young's modulus of steel is not known exactly. We assume

therefore that it's known with a 10% accuracy, and is therefore in the range 180–220 N/m^2. If we assume no correlation between the uncertainty in the elements, we can describe the Young's modulus for each element as

$$E_i = 200 + 20\varepsilon_i \quad \forall i \in \{1,2,3\}.$$

We can then calculate the stress for each element using purely interval computations, or first, second, third, fourth order affine arithmetic. For purely interval computation, we use the state-of-the-art interval FEM method devised by Muhanna and Mullen [**?**], which gives very good bounds despite the high correlation between uncertainties in the matrix A. We consider this the "0th" order affine arithmetic result, in the sense that no correlation is tracked.

element	0-th	1st	2nd	3rd	4th
I	[7.46, 25.99]	[12.01, 20.23]	[12.58, 19.53]	[12.71, 19.38]	[12.72, 19.37]
II	[0, 19.15]	[1.24, 4.21]	[1.85, 3.32]	[1.98, 3.16]	[1.98, 3.15]
III	[6.71, 28.07]	[11.83, 20.45]	[12.55, 19.57]	[12.71, 19.38]	[12.71, 19.37]

Table 1: Stress range for each element

The total range of the stress for each element is show in Table 1 (only two decimal digits are displayed because of space constraints). In Table 2 we show the radius of the stress for each element, and for higher orders we also show the relative gain for the radius: $(r_{\text{lower}} - r_{\text{higher}})/r_{\text{lower}}$.

The first thing that can be noticed is that although higher orders provide tighter ranges, the relative gain in radius decreases exponentially. By contrast, the computational complexity grows as described in section 5.1: the number of additional terms tracked at order k is $\binom{r+k-1}{k}$, where r is the number of original noise symbols ($r = 3$ in our case). The relative cost for first, second, third, fourth order is therefore $3, 3+6 = 9, 9+10 = 19, 19+15 = 34$.

Finally, we remark an expected result: the benefits of using higher orders are more significant for the second finite element than for the other two. Indeed, the middle element's stress depends on all three uncertainties, resulting in more sophisticated correlation for high-order mixed terms (such as $\varepsilon_1^2\varepsilon_2$).

7. Conclusions

Standard AA provides an excellent tool for self-verified computing, with significant benefits over traditional interval analysis. When a large number of nonlinear operations are involved, however, the loss in correlation tracking caused by the appearance of higher order terms can significantly limit the benefit of AA, requiring the use of a higher-order extension.

element	radius	gain
0th order		
I	9.26538	—
II	9.5765	—
III	10.6804	—
1st order		
I	4.11364	55.6%
II	1.48374	84.5%
III	4.31171	59.6%
2nd order		
I	3.47473	15.5%
II	0.734874	50.4%
III	3.51251	18.5%
3rd order		
I	3.33385	4.0%
II	0.591323	19.5%
III	3.33588	5.0%
4th order		
I	3.32721	0.2%
II	0.584707	1.1%
III	3.32759	0.2%

Table 2: Stress radius gain for each element

The extended AA described in this paper works around the limitations of two other extensions (MAA in tensor form and GQF) offering a mathematical structure that can be used to reliably track arbitrary-order correlation on uncertanties withour special algorithmical requirements.

The benefits obtained by using this extension are most significant when MAA in tensor forms becomes unwieldy to manipulate, but quadratic correlation as tracked by GQF is not sufficient to provide optimal bounds.

GIUSEPPE BILOTTA
Dipartimento di Matematica
Università di Catania
e-mail: bilotta@dmi.unict.it